Lec 21: Virtual Memory

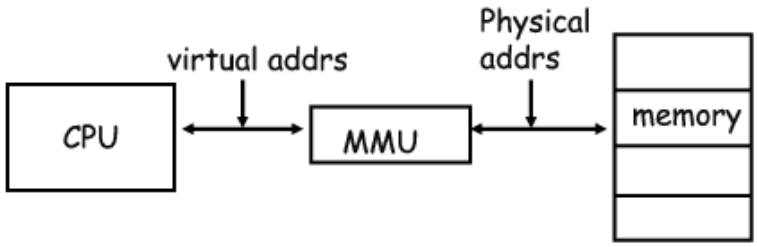
Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

- HW 4 out: smash the stack
  - Due Nov 14th
- PA 3 out Nov 14th
  - Due Nov 25th (feel free to turn it in early)
  - Demos and pizza party: Dec 1st or 2nd
- Prelim 2: Dec 4th
- Final project: distributed multicore ray tracer
  - Due exam week

How to make it work?

- Challenge: Virtual Memory can be slow!
- At run-time: virtual address must be translated to a physical address
- MMU (combination of hardware and software)

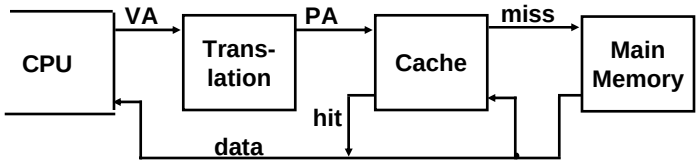


Address Translation

- How to translate addresses?
  - Per word? Much too expensive
  - Per block? Sure, but what is block size?
- Costs dictate granularity of translation
  - Cost to disk is very large
  - Block size has to be large too
    - Amortization
- Page Table: stores this translation
  - Basically a huge array of translations
  - Each process has one

Virtual Addressing with a Cache

- It takes an *extra* memory access to translate a VA to a PA

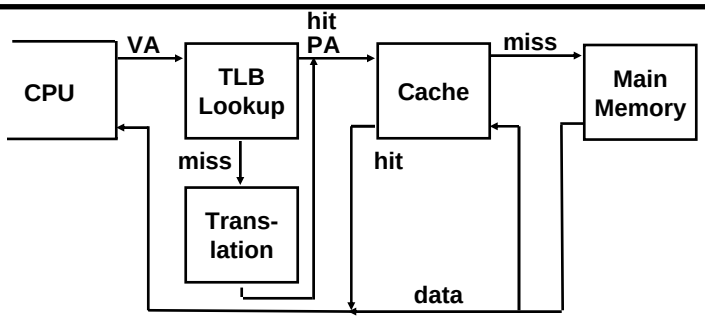


- This makes memory (cache) accesses very expensive (if every access was really *two* accesses)

Translation Lookaside Buffer (TLB)

- Hardware solution to problem
- A small cache of recently used address mappings
  - TLB hit: avoid a page table lookup

A TLB in the Memory Hierarchy



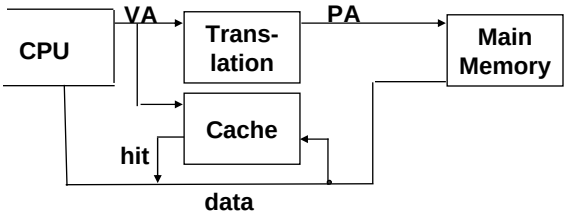
- A TLB miss:
  - If page in main memory, TLB miss can be handled (in hardware or software)
    - Load from the page table into the TLB
    - Takes 10's of cycles

Hardware/Software Boundary

- Translation Lookaside Buffer (TLB) that caches the recent translations
  - TLB access time is part of the cache hit time
  - May allot an extra stage in the pipeline for TLB access

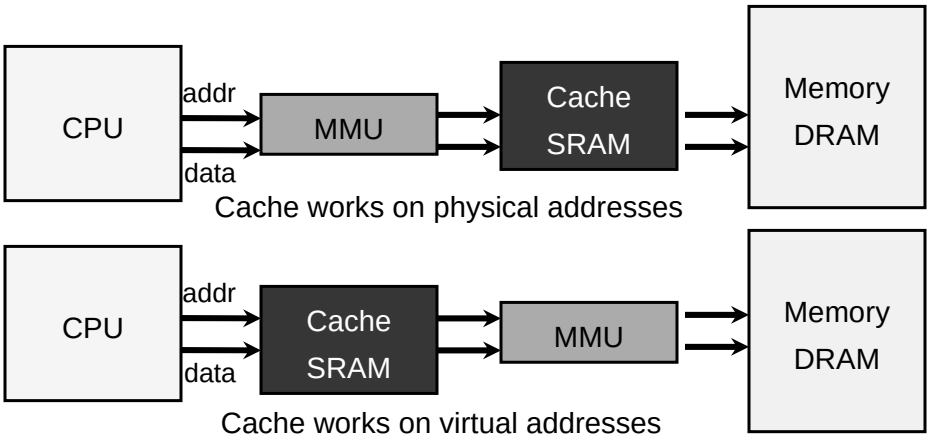
Remove TLB from critical path?

- A virtually addressed cache would only require address translation on cache misses



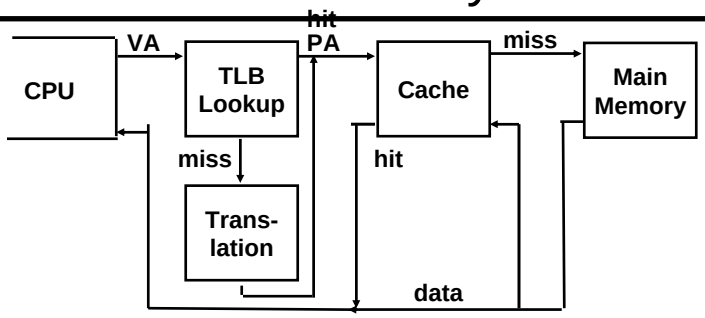
- Cons: have to flush the cache on context switch

Virtual vs. Physical Caches



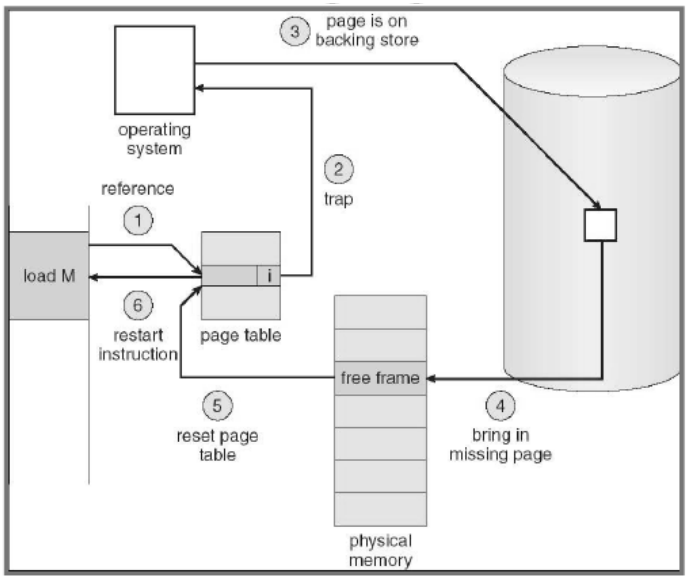
- L1 (on-chip) caches are typically virtual
- L2 (off-chip) caches are typically physical

A TLB in the Memory Hierarchy



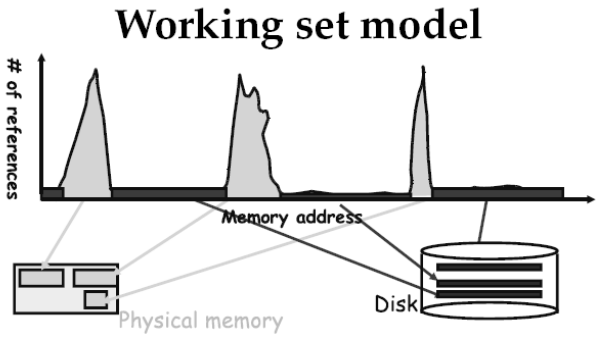
- A TLB miss:
  - If the page is not in main memory
    - Page fault!
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true page faults

Paging



Example: paging to disk

- Compiler (gcc) needs a new page of memory
- Trap/exception into OS
  - OS gets page from application (vi)
- If page is clean, give up page to gcc
- If page is dirty, … only copy in memory
  - Write to disk, before giving it to gcc
- Mark page invalid in vi
  - (if vi needs this soon, it will trap)

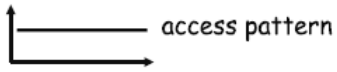


Thrashing

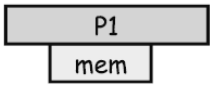
- Thrashing: processes on system require more memory than it has
  - gcc removes vi page
  - vi comes back and removes gcc page
  - Spend lot of time waiting to read pages
  - i/o device at 100% utilization
    - But no useful work is getting done
  - We want Virtual Memory
    - Size of disk, access time of main memory
  - We have
    - Access time of disk

Reasons for Thrashing

- Process doesn't reuse memory
  - Past != future



- Process reuses memory
  - But it doesn't fit



- Individually, processes fit in memory
  - But there are too many